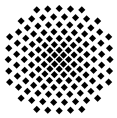

Universität Stuttgart

SimTech
Cluster of Excellence

M. Sonntag D. Karastoyanova

Compensation of Adapted Service Orchestration Logic in BPEL'n'Aspects

Stuttgart, July 2011

Institute of Architecture of Application Systems (IAAS)
University of Stuttgart,
Universitaetsstrasse 38
70569 Stuttgart, Germany
{sonntag, karastoyanova}@iaas.uni-stuttgart.de
http://www.iaas.uni-stuttgart.de

Abstract BPEL'n'Aspects is a non-intrusive mechanism for adaptation of control flow of BPEL processes based on the AOP paradigm. It relies on Web service standards to weave process activities in terms of aspects into BPEL processes. This paper is a logical continuation of the BPEL'n'Aspects approach. Its main objective is to enable compensation of weaved-in Web service invocations (activities) in a straightforward manner. We present (1) requirements on a mechanism for compensation of weaved-in process activities; (2) the corresponding concepts and mechanisms to meet these requirements; (3) an example scenario to show the applicability of the approach; and (4) a prototypical implementation to prove the feasibility of the solution. This work represents an improvement in the applicability of this particular adaptation approach since processes in production need the means to compensate actions that are included into processes as result of an adaptation step, too. The concept is generic and hence can also be used by other approaches that adapt control flow.

Keywords Workflow, Service Composition, BPEL, Compensation, Aspect-orientation, Adaptability.

Reference Sonntag, M. and Karastoyanova, D. (2011) "Compensation of Adapted Service Orchestration Logic in BPEL'n'Aspects." In: Proceedings of the 9th International Conference on Business Process Management (BPM 2011), Clermont-Ferrand, France.

Stuttgart Research Centre for Simulation Technology (SRC SimTech)

SimTech – Cluster of Excellence
Pfaffenwaldring 7a
70569 Stuttgart
publications@simtech.uni-stuttgart.de
www.simtech.uni-stuttgart.de

Compensation of Adapted Service Orchestration Logic in BPEL'n'Aspects

Mirko Sonntag, Dimka Karastoyanova

Institute of Architecture of Application Systems, University of Stuttgart,
70569 Stuttgart, Germany
{sonntag, karastoyanova}@iaas.uni-stuttgart.de

Abstract. BPEL'n'Aspects is a non-intrusive mechanism for adaptation of control flow of BPEL processes based on the AOP paradigm. It relies on Web service standards to weave process activities in terms of aspects into BPEL processes. This paper is a logical continuation of the BPEL'n'Aspects approach. Its main objective is to enable compensation of weaved-in Web service invocations (activities) in a straightforward manner. We present (1) requirements on a mechanism for compensation of weaved-in process activities; (2) the corresponding concepts and mechanisms to meet these requirements; (3) an example scenario to show the applicability of the approach; and (4) a prototypical implementation to prove the feasibility of the solution. This work represents an improvement in the applicability of this particular adaptation approach since processes in production need the means to compensate actions that are included into processes as result of an adaptation step, too. The concept is generic and hence can also be used by other approaches that adapt control flow.

Keywords: Workflow, Service Composition, BPEL, Compensation, Aspect-orientation, Adaptability.

1 Introduction

In the last decade, the Business Process Execution Language (BPEL) [1] became the de facto standard for modeling and execution of service orchestrations in research and industry. BPEL processes rely on Web services (WS) for the implementation of process activities and are themselves published as WSs, which is known as recursive aggregation model. The WSs used by a BPEL process are specified in an abstract way only in terms of their port types (i.e. interfaces) and operations. Concrete WS implementations can be selected using a static service binding strategy [2] or can be chosen at runtime by the supporting middleware, the enterprise service bus (ESB). The latter strategy is called late or dynamic binding of services and allows for substituting WS implementations used in a BPEL process without a need to change the process model itself. A prerequisite for this kind of flexibility (on the functions dimension of processes [3]) is that the interface of services is not changed. In practice, however, interface stability cannot be guaranteed. Thus, in order to react to changed

market conditions or service landscapes, process models need to be modified while their instances are being executed, which may entail complex process instance migration operations.

In our previous work, we proposed an adaptation mechanism for BPEL based on the aspect-oriented programming (AOP) paradigm called BPEL'n'Aspects [4, 5]. It allows dynamic weaving of WS calls into BPEL process models or running instances, which may indeed realize change operations on the business logic (control flow) dimension of processes like deletion, insertion, or skipping of activities or other elements. Since these WS invocations are specified externally as so-called *aspects*, the adapted processes do not need to be modified in any way, which quite in the spirit of AOP renders the approach a non-intrusive one. This is a great practical importance since it can be applied to existing BPEL processes currently in production.

Adapting running processes introduces new challenges for the compensation of (parts of) processes. Compensation is a mechanism to reverse actions performed in a long-running transaction (LRT) [3]. In BPEL, compensation is realized by compensation handlers that are attached to scopes and that specify behavior to undo the work accomplished by the scopes in case of a fault. Compensation is crucial for the applicability of BPEL in real world scenarios where failures cannot be avoided.

Functionality that is weaved into processes at runtime cannot be foreseen and accounted for by compensation handlers that were specified at process design time. This work therefore presents a concept and its implementation for the compensation of dynamically weaved-in aspects into BPEL processes. It is the next logical step to improve the applicability of the BPEL'n'Aspects approach in practice and is the main contribution of this work. In particular, we impose requirements on a mechanism to compensate process tasks that are included into a process during its execution as result of an adaptation step, a concept that satisfies these requirements, an example scenario to demonstrate its feasibility, and a prototypical implementation. The concepts in the paper are demonstrated to be applicable for a BPEL environment; however they can equally be applied for other process meta-models and languages.

The rest of the paper is structured as follows. Section 2 shortly introduces the BPEL'n'Aspects approach, which is needed as basis to understand the remainder of the paper. Section 3 presents the approach for aspect compensation. Section 4 describes the prototype implementing the aspect compensation mechanism. Related work is presented in Section 5. Our conclusions are given in Section 6.

2 BPEL'n'Aspects in a Nutshell

Business processes need to be flexible to allow enterprises to react on changed market conditions and to stay competitive. BPEL provides a built-in flexibility mechanism: used services can be specified in an abstract manner in terms of WSDL port types and operations, i.e. only the type of a service is specified. Concrete services implementing this type can then be chosen with the help of different binding strategies [2]. This kind of flexibility allows BPEL engines to change concrete services even at runtime.

But it is not possible to switch to a service with a different type (i.e. with a different port type and/or operation) during the execution of processes. This can only

be done by modifying the process model appropriately and propagating the changes to running instances via instance migration techniques [6], which is a cumbersome task to perform.

The combination of AOP techniques with the workflow technology [4, 7] allows adapting running workflows without a need to change the workflow model. With AOP crosscutting concerns can be specified in a modular and reusable fashion [8]. New behavior can be weaved into the program logic at specific points while keeping a modular design of the system. Instance migration techniques are not needed. In former work, we presented such a solution called *BPEL'n'Aspects* [4]. In the following, we will summarize its most important features.

2.1 Weaving

The BPEL'n'Aspects approach weaves aspects into BPEL process models and instances. BPEL engine events are used to signal points of interest where aspects can be weaved in. This decouples weaving of aspects from specific engine implementations.

BPEL'n'Aspects relies on existing technologies and standards. WS-Policy [9] is used to specify aspects. Listing 1 shows the structure of aspects as WS-Policy assertion. Association of aspects with BPEL processes is done with WS-PolicyAttachment [10]. This mechanism allows attaching aspects to processes at runtime without changing the target process definition.

```
<a4b:Aspect Id="..."?>
   <a4b:Advice name="..."?>
        <a4b:when type="before|instead|after"/>                    → 1
        <wsa:EndpointReference>...</wsa:EndpointReference>         → 2
        <a4b:Operation name="..."/>                                → 3
        <a4b:InputTransformation>...</a4b:InputTransformation>?    → 4
        <a4b:OutputTransformation>...</a4b:OutputTransformation>?  → 5
   </a4b:Advice>
   <a4b:Pointcut>
        <a4b:ProcessArtifact type="activity|link|..."
                  identifier="..." />*                             → 6
   </a4b:Pointcut>
</a4b:Aspect>
```

Listing 1. Definition of an aspect as WS-Policy assertion.

AOP terms map on BPEL'n'Aspects terms as follows: *joinpoints* are BPEL activities or transition conditions. But in principle every language construct can be a potential joinpoint. *Advices* are WSs specified by an endpoint reference (EPR) and an operation (labels 2 and 3 in Listing 1). *Pointcuts* identify concrete process artifacts where aspects are to be weaved in (label 6). There are three types of advices: an advice can be weaved in *before*, *instead*, or *after* a given joinpoint (label 1). *Aspects* are packages that connect engine events (= pointcut + advice type) with WSs (advice). Input and output data of an advice can be fetched from and written to process variables with the help of input and output transformation operations (labels 4 and 5).

These transformations can also be used to specify data mediation functionality.

BPEL'n'Aspects weaves WS invocations into BPEL processes instead of BPEL code. We consider this solution both more flexible and non-intrusive for the following reasons. First, the called WS may be implemented by a BPEL process or any other mechanism. Second, we do not need to extend the BPEL language (and thus the used BPEL engine) to account for the element related to the aspects. An example of an aspect definition is shown in Listing 3 in Section 3.4.

2.2 Architecture

The BPEL'n'Aspects approach has already been implemented by a prototype presented in [4, 5]. Its architecture consists of four main components (see Fig. 1). Gray components are either newly added or extended in this work and are discussed later in Section 4. These components are connected via pub/sub mechanisms for flexibility and extensibility reasons as well as due to the suitability of the pub/sub paradigm for implementing event-based systems.

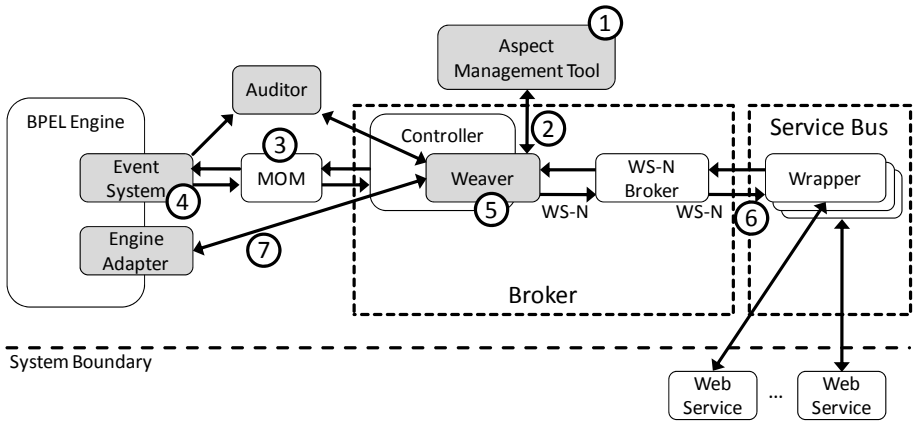

Fig. 1. Architecture of the BPEL'n'Aspects prototype.

With the *aspect management tool*, a workflow designer defines an aspect (1) and deploys it to the *broker* (2). The broker weaves aspects into processes by subscribing to navigation events of the *BPEL engine* (3). The engine is extended by an event system to publish navigation events (the pluggable framework [11]). If an event of interest is signaled (4), the execution of an advice is triggered (5). The broker delegates the advice execution (i.e. the service invocation) to the *service bus* (6). The service bus is realized by wrappers. A wrapper can be seen as a gateway to invoke services. That way the broker is decoupled from WS invocations. Finally, the broker propagates results of a service invocation to the engine with the help of an engine adapter (7). The engine adapter allows accessing process instance data such as variable instances, activity states, etc. One of the main goals of the architecture and prototype was to touch the BPEL engine as little as possible, especially to avoid the migration of process instances. Arbitrary BPEL engines can be used for the

BPEL'n'Aspects approach. The engines should offer an appropriate event system and access on process instance data or may need extensions to provide these functions.

The prototype makes use of following technologies and implementations. The Active XEngine is taken as BPEL engine. Engine extensions are accomplished with the help of AspectJ. The ActiveMQ Java Message Service (JMS) implementation is used as message-oriented middleware (MOM). For the notifications between weaver and bus the WS-Notification (WS-N) implementation WS-Messenger is taken. The aspect management tool is a standalone Java application. The weaver is implemented as J2EE web application with a GUI based on Swing. It uses the Apache Neethi implementation of WS-Policy.

3 Compensation in Adapted Service Orchestrations

Compensation in service orchestrations is a crucial functionality for the applicability of the service composition concept in real world scenarios. In practice, there are many sources of failures and hence faults cannot be avoided. For example, servers can become unavailable due to network errors or software updates that require a restart. Services may be moved to different locations or service interfaces may be modified. It is therefore important to carry out (parts of) business processes within transactions that are able to undo work in case of an error and thus preserve consistency of data. Since business processes are often long-running, transactions with ACID (atomicity, consistency, isolation, durability) properties are not applicable. The isolation constraint can lead to locked data that cannot be accessed by other transactions while the transaction that uses the data runs. Practice shows that this is disadvantageous in scenarios with long-running processes.

Workflow technology therefore relies on compensation-based transaction models that are used to span LRTs. LRTs specify compensating behavior that reverses the effects of already completed work in case of an error. Note that reversing does not mean to rollback finished work but to use operations to revoke the effects of a transaction. For example, booking a hotel room can be compensated by a "cancel hotel room" operation. This often entails a cancellation fee and is therefore not equal to a rollback of an ACID transaction.

The compensating actions are specified during the design time of the process model and are thus available for use in any of the process instances. Compensation of adapted service orchestrations is beyond the scope of conventional compensation mechanisms since the changed functionality cannot be foreseen at process design time. In order to be applicable in practice, concepts for the compensation of adapted service orchestrations need to be devised. This section provides such concepts for the BPEL'n'Aspects approach.

3.1 Compensation in BPEL

In this section we provide some background to the compensation mechanism in BPEL. Compensation in BPEL is achieved by compensation handlers that can be

attached to `scope` activities. Compensation handlers are intended to provide actions to undo the (successful completed) work of a scope. There are two special cases of scopes, namely the process scope and invoke activities that act as implicit scopes. A BPEL process can contain several, possibly nested scopes. The process scope is the outermost scope and thus contains all other scopes. A compensation handler of an implicit scope of an `invoke` activity can be used to define pairs of activities where one activity stands for an action and the other for a compensating action (e.g. "book hotel room" and "cancel hotel room").

If no compensation handler is defined, a default compensation handler is used instead. Default compensation invokes all compensation handlers of the immediately enclosed and successfully completed scopes in reverse execution order. Note that a compensation handler is only installed if its scope is completed. That means only completed scopes can be compensated. A compensation handler must be able to access variables with the content that was valid at completion time of the considered scope. A snapshot of all variables that are visible within a scope is therefore stored when the scope completes.

BPEL provides two ways to invoke compensation handlers, corresponding to explicit and default compensation. Note that compensation handlers can only be invoked from within fault or other compensation handlers of immediately enclosing scopes. First, a `compensate` activity with given `scope` attribute invokes the compensation handler of the named scope. Second, a `compensate` activity without attributes invokes all compensation handlers of immediately enclosed scopes in reverse execution order.

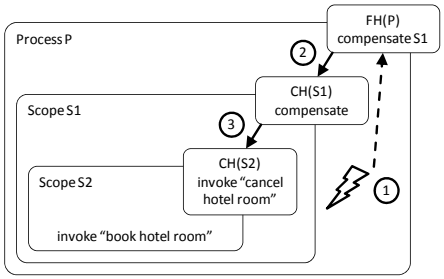

Fig. 2. Fault and compensation handlers in a BPEL process.

Fig. 2 illustrates the compensation in BPEL (version 1.1). Process P contains two nested scopes, S1 and S2. In S2, an `invoke` activity calls a WS to book a hotel room. A fault handler is attached to P and there are compensation handlers for S1 and S2. Assume that S1 and S2 are completed successfully during execution. Their compensation handlers get installed. A fault occurs in process scope P. The fault is propagated to fault handler FH(P) (step 1). FH(P) handles the fault by explicitly invoking the compensation handler CH(S1) of scope S1 by a `compensate` activity with scope attribute (step 2). CH(S1) is designed to simply compensate all immediately enclosed scopes in reverse order by a `compensate` activity without scope attribute. The compensation handler CH(S2) of scope S2 is therefore invoked

(step 3). Finally, CH(2) contains an `invoke` that cancels the booked hotel room. The fault is now handled successfully by the compensation of scopes S1 and S2.

3.2 Requirements on Aspect Compensation

Compensating behavior in a process model is designed along with its normal behavior in the process modeling phase. As outlined earlier, the behavior of aspects usually cannot be foreseen at process modeling time and thus cannot be subject to regular compensation handlers. Hence, specification of an aspect and its compensation must happen in the same phase, namely in the creation phase of an aspect, which is during the deployment or execution phase of processes.

Our idea is to use the same aspect weaving mechanism also for the compensation of aspects [12]. That means we propose to weave compensation aspects into processes to compensate weaved-in aspects. We thereby keep the concept of activity pairs for an action and its compensation action: each aspect can get a compensation aspect attached. Such a compensation aspect refers to the compensation handler of the scope in which the aspect that needs to be compensated is weaved in. In the following we present considerations that influence the concept of aspect compensation.

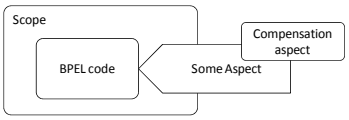

Fig. 1. Weaving an aspect into a scope without compensation handler.

An aspect can be weaved into a scope without explicit compensation handler (see Fig. 3). For such scopes, an implicit compensation handler is installed at process runtime by the BPEL engine. Such an implicit compensation handler contains only a single `compensate` activity that triggers compensation of all immediately contained scopes in reverse execution order. Since the implicit compensation handler is not visible in the process model definition, it is not possible to define a pointcut for a compensation aspect. We therefore define that the advice of a compensation aspect without given pointcut is associated with the compensation handler of the scope in which the aspect that is to be compensated is weaved in. During the execution of the default compensation handler of a scope with a weaved-in aspect, the weaver must use runtime information of the particular process instance to identify the position of the weaved-in functionality. Based on this the weaver is capable of locating the correct position of the compensating aspect in the reverse-order graph for the default compensation. This imposes the requirement on the execution environment to collect information about executed aspects in the audit trail (a component that collects the run time information about all process instances).

An aspect can be weaved into a scope that has an explicit compensation handler (see Fig. 4a). In this case, the compensation aspect must be associated with the specified compensation handler.

We also consider an additional case where an aspect is weaved into a scope that is nested within another scope (see Fig. 4b). The `outer` scope S1 has an explicit

compensation handler CH(S1). The inner scope S2 can have an explicit or default compensation handler. That is the reason why CH(S2) is depicted by dashed lines. An aspect is weaved into scope S2. That means its compensation aspect belongs to CH(S2). If S1 needs to be compensated, CH(S1) is invoked. Now, there can be cases where CH(S1) ignores CH(S2), i.e. the explicit compensation definition does not contain actions involving CH(S2). That means CH(S2) is neither explicitly invoked by a `compensate` activity with S2 as scope name nor implicitly by a `compensate` without specified scope name. In such cases, the compensation of the weaved-in aspect would also be ignored. A mechanism is needed to enforce the compensation of aspects in these kinds of scenarios.

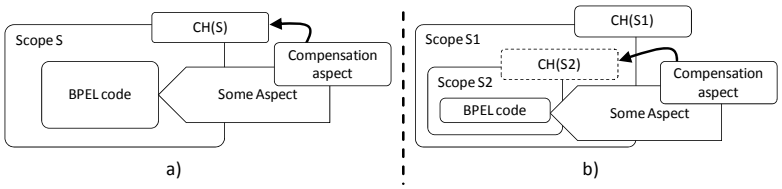

Fig. 4. Weaving an aspect (a) into a scope with explicit compensation handler and (b) into a nested scope.

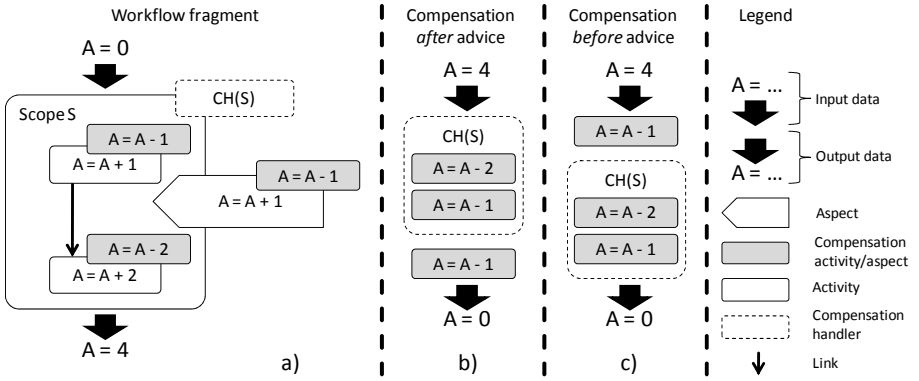

Fig.5. Successful aspect compensation before/after a compensation handler.

In many cases it may be sufficient that a compensation aspect is weaved into a process before, instead, or after a compensation handler. That means the compensation handler is considered a black box. Fig. 5 illustrates this. An aspect is weaved into a mathematical equation that simply adds up some values (a). If scope S must be compensated, it is unimportant whether the aspect is compensated *after* (b) or *before* (c) the compensation handler. The result is correct in both cases (`A = 0`). Note that an *instead* advice would replace a complete compensation handler. Although this may be wished in some scenarios, this would not yield a correct result in the given example (`A = 3`) and is therefore not illustrated.

If the aspect's operation is switched to a multiplication (see Fig. 6a), a case can easily be created where it is insufficient to regard a compensation handler as black

box. Invoking the compensation aspect *after* (b) or *before* (c) the compensation handler yields an incorrect result. Weaving the compensation aspect into the compensation handler is needed (d).

In case several compensation aspects are defined for the same joinpoint with the same advice type, a precedence ordering for their execution is needed. The order should be reverse to the execution order of the aspects that are to be compensated. That means execution timestamps of executed aspects need to be tracked by the execution environment. As outlined above, this requirement is already imposed for a correct default compensation behavior.

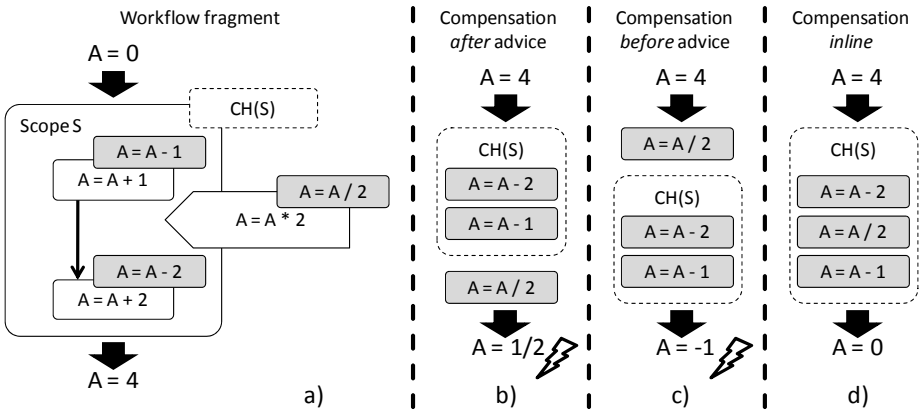

Fig. 6. Successful aspect compensation within a compensation handler.

It is important for the compensation of a scope to access values of variables that were valid at scope completion time. A BPEL engine therefore stores a snapshot of all variables visible in a scope when the scope completes. These snapshots can then be accessed by compensation handlers. Similarly, variable values used or produced by an aspect may be important for its compensation. A snapshot of all variables visible for an aspect must therefore be stored after aspect completion. These values must be accessed by the weaver during aspect compensation.

3.3 Realization of the Aspect Compensation Concept

In order to realize the concept of aspect compensation, we extend the WS-Policy assertion specification for aspects (see Listing 2) [12]. The definition of a compensation aspect for an aspect is done by reference with the help of an aspect identifier. This allows decoupling of aspect and compensation aspect and thus fosters reusability. The new element `CompensationAspect` is used within an aspect to point to an aspect that implements compensating behavior (3). Definition of a compensation aspect is optional for two reasons. First, not each aspect may need a compensation aspect. Second, a compensation aspect is not allowed to reference another compensation aspect because this would mean a compensation of the compensation.

The new `compensating` attribute (1) of an advice is used to mark an aspect as compensation aspect. Note that compensation aspects cannot be weaved into normal process behavior but are always associated with the compensation handler of the scope of the aspect to be compensated.

As outlined in Section 3.2 compensation handlers in nested scopes may be overridden. Thus, the compensation of weaved-in aspects may be skipped as a side-effect. With the `alwaysCompensate` attribute (2) compensation of an aspect can be enforced even if its compensation handler may be ignored. In this case, the advice type and pointcut are irrelevant as they refer to an overridden compensation handler. The compensation aspect is therefore executed after the compensation handler of the enclosing scope.

```
<a4b:Aspect Id="..."?>
   <a4b:Advice name="..."?
            compensating="true|false"                    → 1
            alwaysCompensate="true|false"?>...           → 2
   </a4b:Advice>
   <a4b:CompensationAspect aspectId="..."/>?              → 3
   <a4b:Pointcut>?                                       → 4
       <a4b:ProcessArtifact type="activity|link|..."
                identifier="..." />*
   </a4b:Pointcut>
</a4b:Aspect>
```

Listing 2. Extended aspect definition.

Finally, we extended the definition of pointcuts by making pointcuts optional (4). This extension only applies to compensation aspects. There are cases where the specification of a pointcut in a compensation aspect is not possible or not desired. If a compensation aspect belongs to a default compensation handler, it is not possible to specify a pointcut that refers to the handler. In this case, the compensation aspect needs to be executed at the correct position of the reverse-order graph. Hence, the specified advice type is unimportant. But even if a compensation aspect belongs to an explicit compensation handler, the pointcut may be omitted. The advice is then applied to the explicit compensation handler and is executed before, after, or instead of it. This also means that a compensation handler can be completely replaced by a compensation aspect if an *instead* advice is used. However, if a pointcut is specified for a compensation aspect, it must refer to an activity/link within the explicit compensation handler that belongs to the scope of the aspect that is compensated.

3.4 Scenario

Imagine an enterprise that operates an online book store. Customers can order books via a WS provided by the book store. The WS is implemented by a BPEL 1.1 process. The process is given in Fig. 7 in BPMN notation. Note that it is not modeled in an optimal way in order to illustrate aspect compensation by default and explicit compensation handlers as well as in nested scopes.

The process is triggered by a book order of a customer. The two main parts of the process are billing and shipment which are executed in parallel. For shipment the books are prepared, i.e. packed, before a third party company is contracted to deliver the books. There are two explicit compensation handlers: book preparation can be undone by putting the books back into the warehouse; shipping is compensated by cancelling the transportation request and putting the books back into the warehouse. The billing path contains the calculation of the bill and the invocation of a service that submits the bill to the customer. There is no explicit compensation handler to undo billing the customer. If billing and shipment are successful, the customer gets a positive notification about his order. A fault handler at the process scope ensures that the process is not aborted in case of an error. Instead, the completed work is undone and the customer is informed that the order could not be handled.

In order to prove the concept of aspect compensation we applied it in three scenarios in which the process is adapted and compensation aspects come into play.

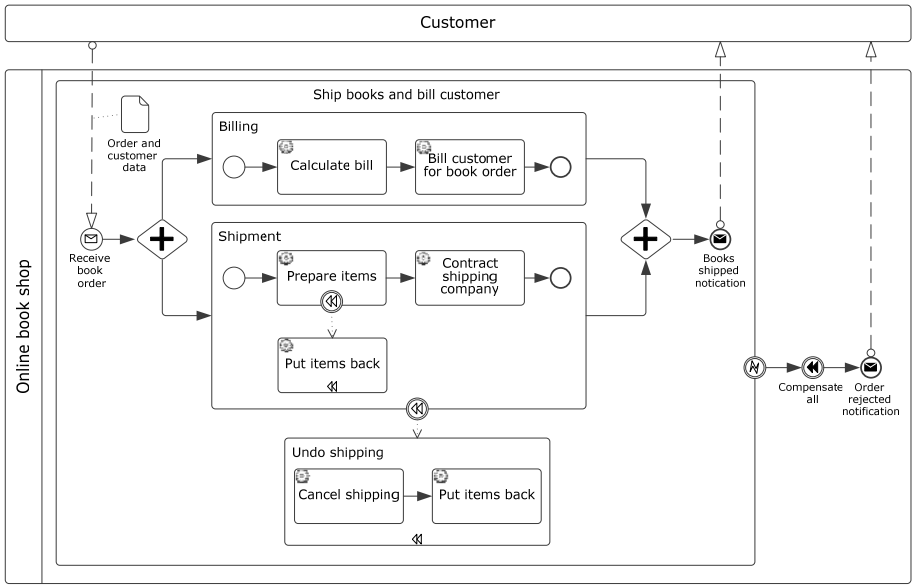

Fig. 7. Business process example for an online book store.

Scenario 1: Credit point program

A credit point program can be easily implemented with an advice weaved-in after billing the customer (Listing 3). The advice *CustomerReward* invokes a WS operation that calculates the number of credit points for the customer depending on the amount of the bill (`rewardCustomer`). In case of an error the granted credit points have to be subtracted. A compensation aspect *CancelCustomerReward* is defined that invokes a WS operation for credit point subtraction (`cancelCustomerReward`). The process does not foresee a compensation handler for the billing scope. That means that a default compensation handler is installed at scope completion time and can be invoked by the higher-level fault handler. The compensation aspect does not specify a pointcut and hence refers to the (default) compensation handler of the billing scope.

The advice type of the compensation aspect is irrelevant since it is part of the reverse-order graph of default compensation behavior.

```
<a4b:Aspect xmlns:a4b="... " id="ns1:CustomerReward">
   <a4b:Advice name="CustomerReward" alwaysCompensate="false"
            compensating="false">
        <a4b:When type="After"/>
        <wsa:EndpointReference ...>...</wsa:EndpointReference>
        <a4b:Operation name="rewardCustomer" suppressFault="false"
                suppressResult="false"/>
        <a4b:InputTransformation>billingRequest
        </a4b:InputTransformation>
        <a4b:OutputTransformation>billingResponse
        </a4b:OutputTransformation>
   </a4b:Advice>
   <a4b:CompensationAspect aspectId="ns1:CancelCustomerReward"/>
   <a4b:Pointcut>
        <a4b:ProcessArtifact type="Activity"
                identifier="//invoke[@name='InvokeBillingService']"/>
   </a4b:Pointcut>
</a4b:Aspect>
<a4b:Aspect xmlns:a4b="... " id="ns1:CancelCustomerReward">
   <a4b:Advice name="CancelCustomerReward" alwaysCompensate="false"
            compensating="true">
        <a4b:When type="Before"/>
        <wsa:EndpointReference ...>...</wsa:EndpointReference>
        <a4b:Operation name="cancelCustomerReward"
                suppressFault="false" suppressResult="false"/>
        <a4b:InputTransformation>billingRequest
        </a4b:InputTransformation>
        <a4b:OutputTransformation>billingResponse
        </a4b:OutputTransformation>
   </a4b:Advice>
</a4b:Aspect>
```

Listing 3. Aspect for credit point program and its compensation aspect.

Scenario 2: Express delivery
Invoking the shipping company is hard-coded into the process in terms of a port type and an operation. In this scenario, we want to make use of another shipping company for express delivery. Imagine our favorite express delivery company offers a WS with a different port type/operation. An aspect with *instead* advice can be used to replace the old `invoke` activity. Accordingly, the compensation of the shipping request needs to be adapted. A compensation aspect is specified that invokes a cancel shipping service of the express delivery company. The pointcut of the compensation aspect points to the “cancel shipping” activity in the compensation handler of the shipment scope. In order to replace the old cancel operation the advice type is set to *instead*.

Scenario 3: Christmas present campaign
The book store plans a Christmas present campaign. Each customer gets a present that is put into the package with ordered books. This can be realized by an aspect that is weaved into the process as *after* advice after the “Prepare items” activity. Of course, this influences the compensation of this activity since both the order items and the present need to be put back to the warehouse. A compensation aspect is therefore

specified that takes the present back to the warehouse. It points to the “Put items back” activity of the compensation handler of activity “Prepare items”. Since the present insertion aspect is executed after the activity, the compensation needs to be performed in reverse order. Hence, the compensation aspect gets a *before* advice. In case the shipment scope completed successfully but the billing failed, the shipment scope is compensated with the help of its compensation handler “Undo shipping”. Unfortunately, this compensation handler overrides the compensation handler of activity “Prepare items” and hence the compensation aspect. Nevertheless, compensation of the aspect is enforced by setting the `alwaysCompensate` attribute to `true`. That means the aspect is compensated although its compensation handler is skipped. Note that the advice type and pointcut of the compensation aspect are ignored since they point to the overridden compensation handler. The compensation aspect is therefore executed after the compensation handler “Undo shipping”.

4 Prototype

We extended the existing BPEL’n’Aspects prototype in order to implement the proposed concepts of aspect compensation (Fig. 1) [12]. The *event system* is extended by events needed to weave compensation aspects into compensation handler: events that are published if a compensation handler is ready for execution, is executing, or has finished execution. A new `variable_changed` event signals that the value of a variable was edited which is needed by the new auditor component.

The *engine adapter* is enhanced with functionality to dynamically register blocking events. That way a process instance can be requested to suspend execution and wait for a resume event. If an aspect was successfully weaved in, the event that triggers its compensation can be registered as blocking at runtime. If a deployed aspect was not weaved in, it does not need to be compensated and no event needs to be registered as blocking. This could, e.g., be the case for an aspect weaved into one of two alternative paths in the process model and when the path without aspect is taken in a particular instance. Dynamically registering blocking events improves efficiency of the solution.

The *auditor* is a new component that tracks and persistently stores event messages exchanged between the engine and the weaver. It serves the role of the so-called audit trail. Since the architecture heavily relies on the pub/sub paradigm, the auditor can simply be connected as an additional listener on the event topic of the engine and the weaver topic of the weaver. The event `variable_changed` of the engine is most important for aspect compensation since it is used by the auditor to store the snapshot of aspect variables. These snapshots and other data can be fetched out of the audit trail via a WS interface provided by the auditor. Furthermore the auditor stores execution timestamps of activities and weaved-in aspects. This information is important for the reverse-order graph of default compensation and for the precedence order of compensation aspects with the same joinpoint and advice type.

The *weaver* is extended with capabilities to identify compensation aspects of aspects and to register them for the compensation of scopes. It can access variable values (i.e. snapshots) needed for the compensation of an aspect over the WS interface of the auditor. Result values of compensating WS calls are forwarded to the

BPEL engine. Additionally, as mentioned earlier, the weaver can register blocking events in the engine over the engine adapter to weave in compensation aspects. Currently, the implementation of default compensation behavior is restricted as follows: compensation aspects are executed after default compensation handlers, i.e. they are inserted at the end of the reverse-order graph. In future, the correct position for compensation aspects needs to be derived from the execution order of activities and aspects of a scope.

The *aspect management tool* is enhanced with the functionality to specify aspects as compensation aspects and to reference them from within ordinary aspects. For more information on the prototype the reader is referred to [12] and [13].

5 Related Work

Much research is done in the application of AOP techniques to service compositions. AO4BPEL [7] is a BPEL extension that allows aspects to be weaved into BPEL processes at runtime. All activities can be considered as joinpoints. As opposed to our approach, pointcuts are modeled with the help of XPath expressions. Advices are either BPEL code or Java method calls. The compensation of aspects is also discussed in AO4BPEL [14]. Scopes are extended to contain a list of compensation handlers (instead of only a single one). If an advice with attached compensation handler is weaved into a process, this compensation handler is added to the list of compensation handlers of the scope the joinpoint belongs to. The execution sequence of the list of compensation handlers is thereby arbitrary. But as we have outlined in Section 3.3, the sequence of compensating actions is of utmost importance.

Weaving aspects into composite telecommunication services is presented in [15]. The approach is used to separate and decouple non-functional requirements such as billing, logging, monitoring, and measurement of service quality from the functional behavior of service orchestrations. Main goals are to improve reusability of services and decrease costs for composite service design and modification. Aspects are specified by a tailor-made language and are weaved in dynamically at runtime. The concept has a limited means to support adaptation mechanisms of orchestrations. For example, an *instead* advice type is not foreseen and hence functionality of a composite services cannot be replaced (as is possible in BPEL'n'Aspects). Advice services can be abstractly specified. Concrete advices are then late-bound at runtime. To the best of our knowledge, the used composition language does not accommodate for the compensation of already completed work. Compensation of weaved-in aspects is therefore not dealt with either.

In [16] AOP is harnessed for two purposes. First, so-called engine aspects are used to extend a BPEL engine with new functionality such as debugging or new language elements. In this case, advices are Java code. Second, with the help of process aspects BPEL code can be dynamically weaved into BPEL processes or instances. It is possible to insert or replace compensation handlers in processes with the help of an aspect. This functionality could be used to compensate weaved in aspects. However, this would be cumbersome and by far not straight forward because compensation handlers would need to be re-defined in order to compensate aspects in scopes with

existing compensation handler. In contrast, our approach with pairs of aspects and compensating aspects allows intuitive compensation of aspects. Replacement of a compensation handler is only one possible scenario.

Another approach that employs AOP mechanisms for the monitoring and supervision of processes is presented in [17]. BPEL processes are annotated with rules in order to control their execution. Weaving of aspects is not conducted dynamically at runtime. A pre-processor is used for static weaving at deployment time. In [18] the authors extend their work towards the recovery of BPEL processes in order to create self-healing systems. The approach implements forward recovery mechanisms with the help of special business rules (e.g. retry of WS invocations). These rules are activated when weaved-in monitoring aspects signal out-of-line situations. Backward recovery (i.e. compensation) in general and compensation of weaved-in aspects in particular is not dealt with by the approach.

6 Conclusions

BPEL'n'Aspects is an approach to improve the flexibility of service orchestrations. It makes use of AOP techniques and existing WSs to dynamically adapt the control flow logic of BPEL processes and instances by inserting, deleting and skipping one or more activities, or other process constructs. Adapting processes at runtime has implications on the compensation behavior of processes: dynamically weaved-in aspects, i.e. dynamically inserted activities, are inherently not a subject to compensation behavior modeled at design time of processes.

In this paper, we extended the existing BPEL'n'Aspects approach to enable compensation of weaved-in aspects, which is the logical continuation of our previous work. We outlined and explained the requirements on such a compensation mechanism: the need for compensation aspects to allow compensation of weaved-in aspects, aspect compensation by default and explicit compensation handlers, aspect compensation in nested scopes, considering explicit compensation handlers as black box for aspect compensation, and weaving compensation aspects into compensation handlers. Based on these requirements we developed a concept for the compensation of aspects. It extends the existing aspect definition with a pointer to a compensation aspect and with the ability to mark an aspect as compensating. A prototypical implementation of an infrastructure enabling these concepts and an example scenario of an online book store prove the feasibility of the solution. Our concepts improve the applicability of the approach in real world situations where faults cannot be avoided and compensation of process steps is an important feature.

In future we intend to apply the BPEL'n'Aspects approach with its novel compensation mechanisms in the field of scientific workflows. It promises to improve adaptability and robustness of usually long-running scientific experiments and simulations. Further research is needed towards the adaptation of processes in stateful environments. Another unsolved issue is a concept to track dynamic changes of processes in order to ensure reproducibility of scientific results.

Acknowledgements. The authors would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology[1] (EXC 310/1) at the University of Stuttgart.

References

1. OASIS: Web Services Business Process Execution Language Version 2.0. OASIS Standard (2007) [online] http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html
2. Weerawarana, S., Curbera, F., Leymann, F., Storey, T., Ferguson, D.F.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More. Prentice Hall (2005)
3. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall (2000)
4. Karastoyanova, D., Leymann, F.: BPEL'n'Aspects: Adapting Service Orchestration Logic. In: Proceedings of the 7th International Conference on Web Services (ICWS 2009)
5. Schroth, R.: Konzeption und Entwicklung einer AOP-fähigen BPEL Engine und eines Aspect-Weavers für BPEL Prozesse. Diploma Thesis No. 2523, University of Stuttgart (2006)
6. Reichert, M., Dadam, P.: $Adept_{flex}$ – Supporting Dynamic Changes of Workflows Without Losing Control. In: Journal of Intelligent Information Systems 10(2) (1998)
7. Charfi, A., Mezini, M.: Aspect-Oriented Web Service Composition. In: Proceedings of ECOWS (2004).
8. Kiczales, G.: Aspect-Oriented Programming. In: Proceedings of ECOOP'97, Finland, 1997.
9. W3C: Web Services Policy 1.5 – Framework. W3C Recommendation (2007) [online] http://www.w3.org/TR/2007/REC-ws-policy-20070904/
10. W3C: Web Services Policy 1.5 – Attachment. W3C Recommendation (2007) [online] http://www.w3.org/TR/2007/REC-ws-policy-attach-20070904/
11. Khalaf, R., Karastoyanova, D., Leymann, F.: Pluggable Framework for Enabling the Execution of Extended BPEL Behavior. In: Proceedings of the 3rd International Workshop on Engineering Service-Oriented Application (WESOA'2007).
12. Wiselka, M.: Erweiterung einer AOP-fähigen BPEL Engine um die Kompensation von eingewobenen Aktivitäten. Diploma Thesis No. 2905, University of Stuttgart (2009)
13. Sonntag, M., Karastoyanova, D.: BPEL'n'Aspects And Compensation: Adapted Service Orchestration Logic and its Compensation Using Aspects. In: Weske, Mathias (Eds.); Yang, Jian (Eds.); Maglio, Paul (Eds.); Fantinato, Marcelo (Eds.): Proceedings of the 8th Int. Conf. on Service-Oriented Computing (ICSOC 2010), Demo Track, 2010.
14. Charfi, A.: Aspect-Oriented Workflow Languages: AO4BPEL and Applications, Fachbereich Informatik, TU Darmstadt, PhD Thesis (2007)
15. Niemöller, J., Levenshteyn, R., Freiter, E., Vandikas, K., Quinet, R., Fikouras, I.: Aspect Orientation for Composite Services in the Telecommunication Domain. In: Proceedings of 7th International Joint Conference ICSOC-Service Wave (2009)
16. Courbis, C., Finkelstein, A.: Towards Aspect Weaving Applications. In: Proceedings of ICSE (2005)
17. Baresi, L., Guinea, S.: Towards Dynamic Monitoring of WS-BPEL Processes. In: Proceedings of ICSOC (2005)
18. Baresi, L., Guinea, S.: A Dynamic and Reactive Approach to the Supervision of BPEL Processes. In: Proceedings of the 1st India Software Engineering Conference (ISEC), 2008.

[1] http://www.simtech.uni-stuttgart.de